DSA5208 - SCALABLE DISTRIBUTED COMPUTING FOR DATA SCIENCE

NATIONAL UNIVERSITY OF SINGAPORE

FACULTY OF SCIENCE

Project 2: Machine Learning on Weather Data

Authors:
Jiayu Shen (ID: A0329922X)
Lujianing Yang (ID: A0329883L)
Keyi Zhao (ID: A0211208X)

1 Introduction

This report details the development and evaluation of machine learning models designed to predict air temperature using the global hourly weather dataset from the National Oceanic and Atmospheric Administration (NOAA). Leveraging the distributed computing power of Apache Spark, this project tackles the challenges of processing and analyzing large-scale meteorological data to build robust predictive models.

The primary objectives of this project are as follows:

- To implement a scalable data processing pipeline for cleaning and transforming massive raw weather observation data.
- To engineer meaningful features from temporal and geographical data to improve model accuracy.
- To train and evaluate at least two distinct machine learning models: Linear Regression and Random Forest Regressor.
- To employ a systematic model selection process using cross-validation to identify the optimal model and its hyperparameters.
- To document and analyze advanced optimization techniques used to ensure performance and fault tolerance during the long-running training process.

2 Methodology

2.1 Computing Environment

All data processing, feature engineering, and model training tasks for this project were executed on a Google Cloud Dataproc cluster. The cluster was provisioned in the us-central1 region using the image version 2.2-debian12. This image provided the core software stack, including:

- **Apache Spark**: 3.5.3
- **Apache Hadoop**: 3.3.6
- **Python**: 3.11

The cluster hardware configuration was defined as follows:

- **Master Node**: 1x `n4-standard-4` (16 vCPUs, 64 GB memory) with a 100GB boot disk.
- **Worker Nodes**: 2x `n4-standard-8` (32 vCPUs, 128 GB memory total) each with a 100GB boot disk.

The JUPYTER optional component was enabled to facilitate interactive analysis and development via JupyterLab.

2.2 Data Preprocessing

The raw dataset presented significant challenges due to its size and complex format. Our preprocessing pipeline, executed in PySpark, was designed to address these issues systematically.
Our entire data cleaning strategy was based on the official NOAA documentation (isd-format-document.pdf), which provided the necessary context for data formats, scaling factors, missing value codes (e.g., 9999, 99999), and quality flags.

2.2.1 Distributed Data Loading

To manage the dataset's scale, we read the complete collection of CSV files for the year 2024 directly from a Google Cloud Storage (GCS) bucket into a Spark DataFrame. This approach allowed Spark to partition and process the data in a distributed manner without overwhelming a single machine's memory.

2.2.2 Specialized Field Parsing

The data required several distinct parsing functions to handle its complex string encoding:

- **Value-Quality Parsing:** Key variables like temperature (`TMP`), dew point (`DEW`), and sea-level pressure (`SLP`) were encoded as `"value,quality"`. We implemented a function (`parse_value_quality`) to split these strings, extract the numerical value, apply the correct scaling factor (e.g., dividing by 10.0), and separate the quality code into its own column.
- **Wind Data Parsing:** Wind (`WND`) data used a more complex 5-part format (direction, quality, type, speed, quality). A separate function (`parse_wind`) was required to split this string, extracting and scaling only the relevant wind speed (item 3) and direction (item 0).

2.2.3 Missing Value Handling

We identified that the dataset uses specific numerical codes (e.g., 9999, 99999) to represent missing data. Our pipeline systematically replaced these codes with 'null' values to ensure they were correctly ignored during statistical calculations and model training.Furthermore, critical geographical features like `LATITUDE` and `ELEVATION`, which were initially loaded as strings, were explicitly cast to `DoubleType` (in Code Cell 5) to ensure they were suitable for modeling.

2.2.4 Quality Control Filtering

To ensure the integrity of our target variable, we filtered the dataset to retain only air temperature readings ('TMP_value') with high-confidence quality flags (codes 0, 1, 4, 5). We also applied a sanity check, filtering out extreme temperature values outside a plausible range of -60°C to 60°C.

2.3 Feature Engineering and Selection

Following the cleaning phase, we engineered a set of predictive features and performed imputation for missing values.

2.3.1 Feature Creation

We derived several new features to enhance the models' predictive power. This included extracting temporal features like 'hour' and 'month' from the timestamp. Crucially, we applied a cyclical transformation to the 'hour' feature using sine and cosine functions to properly model its periodic nature of 24 hours. Additionally, we engineered `abs_latitude` from `LATITUDE` to treat both hemispheres similarly, and `temp_dew_spread` (the difference between `TMP_value` and `DEW_value`) as potential predictors (Code Cell 3).

2.3.2 Feature Imputation

To handle missing values in our selected features, we employed simple yet effective imputation strategies (Code Cell 4). For missing `DEW_value`, we created `dew_final` by imputing nulls with the value of `TMP_value - 5`. For missing `wind_speed`, we calculated the median value from the entire dataset (using `approxQuantile`) and used this to fill nulls, creating the `wind_speed_final` column.

2.3.3 Final Feature Selection

After experimentation, we selected a final set of 7 features for our models: `dew_final`, `hour_sin`, `hour_cos`, `month`, `LATITUDE`, `ELEVATION`, and `wind_speed_final`. These features were chosen to provide a comprehensive view of the conditions influencing temperature while avoiding data leakage (e.g., by not using `temp_dew_spread` alongside `dew_final`).

3 Model Training and Validation

Our modeling process was designed for robust evaluation and hyperparameter optimization.

3.0.1 Data Splitting

The fully processed dataset was randomly split into a training set (70%) and a test set (30%). This standard practice ensures that our final model evaluation is performed on unseen data.

3.0.2 Feature Pipeline

We constructed a `pyspark.ml.Pipeline` to standardize our feature preparation process (Code Cell 6). This pipeline consisted of two stages: a `VectorAssembler` to combine all 7

selected features into a single `assembled_features` vector, and a `StandardScaler` to scale the features to have zero mean and unit variance.
This feature pipeline was fitted once on the training data. Then, both the training and test sets were transformed. Crucially, we immediately called `.cache()` on both the resulting `train_featured` and `test_featured` DataFrames. We then triggered this cache by calling `.count()`, forcing Spark to store the fully processed feature sets in memory. This step was essential to prevent re-computation during the multiple iterations of cross-validation.

3.0.3 Model Selection and Tuning

We initially considered three models: Linear Regression, Random Forest Regressor, and Gradient Boosted Trees (GBT). However, preliminary tests on a local, smaller dataset revealed that GBT was significantly slower than the other two models. Given its high computational cost and marginal performance improvement over Random Forest on the small data, we decided to exclude GBT from the final large-scale training to manage project costs and time (Code Cell 9, commented).
Our final evaluation (Code Cells 7 & 8) focused on **Linear Regression** (LR) as an efficient baseline and **Random Forest Regressor** (RF) for its ability to capture non-linearities.
For both models, we used a `CrossValidator` with 3 folds and `RegressionEvaluator` (metric: RMSE) to perform hyperparameter tuning. The parameter grids were:

- **Linear Regression:** `regParam` ([0.001, 0.01, 0.1]), `elasticNetParam` ([0.0, 0.5, 1.0]), and `maxIter` ([100, 200]).
- **Random Forest:** `numTrees` ([30, 50]) and `maxDepth` ([10, 15]).

The model with the lowest average RMSE across the folds was selected as the best.

4 Results and Evaluation

This section presents the quantitative results of our model training and provides a detailed analysis of the best-performing model.

4.1 Model Performance Comparison

Both models were tuned via 3-fold cross-validation and evaluated on the held-out test set. The performance metrics, including RMSE, R-squared (R^2), and Mean Absolute Error (MAE), are summarized in Table 1.

Table 1: Final Model Performance on the Test Set

Model	Test RMSE (°C)	Test R^2	Test MAE (°C)	Training Time (s)
Linear Regression	5.4948	0.7926	3.7965	1339.2
Random Forest	**3.7826**	**0.9017**	**2.7040**	**41524.5**

The results clearly show that the Random Forest model substantially outperformed the Linear Regression baseline. It achieved an RMSE of $3.78°C$, representing a 31.1% reduction in error compared to the Linear Regression model. Its R^2 value of 0.9017 indicates that our Random Forest model successfully explains 90.2% of the variance in air temperature. This superior performance came at a significant computational cost. The Random Forest training ran for approximately 11.5 hours (41,524.5 seconds), whereas the Linear Regression finished in about 22 minutes (1339.2 seconds). The original output for the RF training cell (Code Cell 8) was cleared prior to submission, as the lengthy process generated a high volume of Spark warnings and a network disconnection prevented the final output from being captured. However, because we implemented model persistence (Code Cell 8, `.save("rf_cv_model")`), we were able to reload the fully trained model in a separate step (Code Cell 8.5) and precisely regenerate the final performance metrics on the test set. This validated our fault-tolerant approach and the importance of saving model artifacts.

4.2 Analysis of the Best Model: Random Forest

Given its superior accuracy, we selected the Random Forest as our final model and conducted a deeper analysis of its characteristics.**Best Hyperparameters:** The cross-validation for Linear Regression selected `regParam=0.001` and `elasticNetParam=0.0`, effectively choosing a Ridge Regression variant. By reloading the saved `CrossValidatorModel`, we identified the best Random Forest model parameters (from Code Cell 8.5 and model metadata) as `numTrees=50` and `maxDepth=15`.

We visualized the Random Forest model's performance on a 50,000-sample subset of the test data to gain qualitative insights (Figure 1).

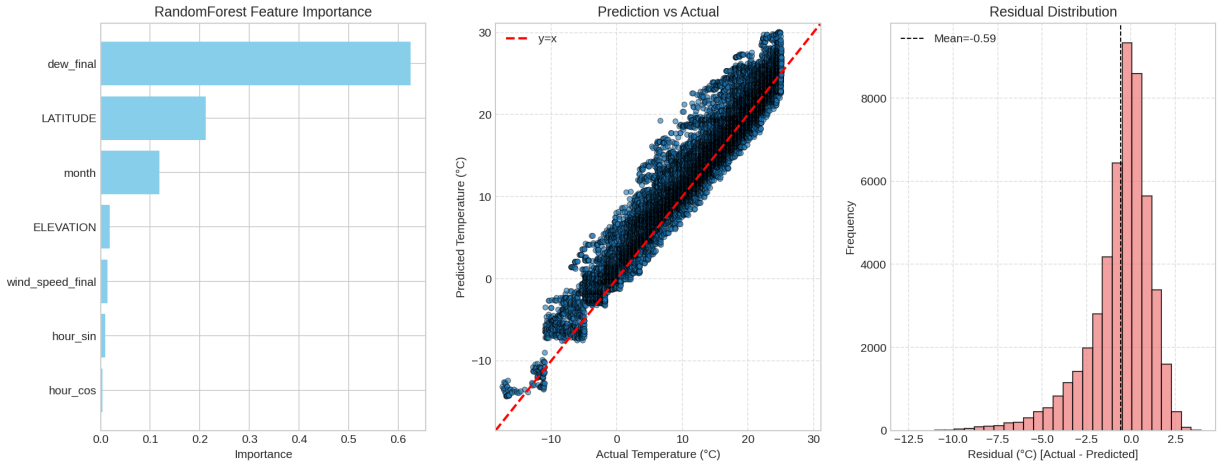

Figure 1: Diagnostic Plots for the Best Performing Model (Random Forest)

The analysis reveals several key points:

- **Feature Importance (Left)**: The feature importance plot reveals that `dew_final` (dew point) is by far the most influential predictor, accounting for over 62% of the

model's decision-making weight. This aligns with meteorological principles, as dew point is intrinsically linked to air temperature. `LATITUDE` and `month` are the next most important, capturing the strong influence of geographical location and seasonality on temperature. Features like `ELEVATION`, `wind_speed_final`, and the cyclical hour encodings (`hour_sin`, `hour_cos`) contribute but are less critical.

- **Prediction vs. Actual (Center)**: This scatter plot shows a strong positive correlation between the predicted and actual temperatures. The data points cluster tightly around the ideal 'y=x' line (red dashed line), visually confirming the high R^2 value. The model performs well across the entire temperature range, from below -10°C to nearly 30°C.

- **Residual Distribution (Right)**: The histogram of residuals (Actual - Predicted) is approximately normally distributed and centered close to zero (Mean = -0.59°C). This indicates that the model's errors are not systematically biased; it does not consistently over- or under-predict. The majority of prediction errors fall within a narrow range of ±2.5°C, which is a strong indicator of a well-calibrated model.

5 Additional Efforts for Performance and Robustness

To handle the large-scale meteorological dataset efficiently and ensure model reliability, we implemented several key optimizations beyond the standard machine learning workflow.

5.1 Strategic Data Caching with Forced Materialization

Recognizing that the `CrossValidator` would perform multiple iterations over the training data, we implemented a targeted caching strategy. Instead of caching the raw data, we placed the `.cache()` operations after all feature engineering steps were completed. More importantly, we immediately called `.count()` on both the training and test DataFrames to force Spark to materialize the results into memory. This approach ensured that all subsequent model training iterations read from memory rather than recomputing the entire feature engineering pipeline, which was critical for the 11.5-hour Random Forest training.

5.2 Optimized Data Processing for Meteorological Data

The raw NOAA data required specialized parsing logic. Key challenges we addressed included:

- **Complex String Parsing:** Meteorological observations like wind data (`WND`) are stored in comma-separated formats (e.g., "direction,quality,type,speed,quality") requiring precise splitting and type conversion.

- **Missing Value Handling:** Proper interpretation of meteorological missing value codes (e.g., 999, 9999, 99999) and appropriate null replacement.

- **Quality Control:** Filtering based on built-in quality control flags to retain only reliable temperature measurements (quality codes 0, 1, 4, 5).

5.3 Memory-Efficient Feature Engineering

Given the dataset size of over 85 million training samples, we implemented several memory optimization techniques:

- **Precise Type Casting:** We explicitly converted columns like `LATITUDE` and `ELEVATION` to `DoubleType()` only when necessary, avoiding Spark's automatic type inference overhead.

- **Cyclical Encoding:** We used sine/cosine transformations for hour data instead of one-hot encoding, reducing feature dimensionality while preserving temporal relationships.

- **Careful Feature Selection:** We selected only 7 core features to minimize memory usage during model training while maintaining predictive power.

5.4 Robust Model Persistence for Fault Tolerance

The extensive training time for the Random Forest model necessitated a fault-tolerant approach. We implemented model persistence by saving the trained `CrossValidatorModel` immediately after training completion. This proved crucial when a kernel crash occurred during visualization—we were able to reload the pre-trained model and complete the analysis without retraining, saving significant computational resources. These optimizations

collectively ensured that the project could scale to handle the 120+ million record dataset efficiently while producing reliable, production-ready temperature prediction models.

6 Conclusion

In this project, we successfully built and evaluated machine learning models to predict global air temperature on a massive dataset. By leveraging Apache Spark, we engineered a robust and scalable pipeline that handled complex data cleaning, feature engineering, and distributed model training. Our findings demonstrate that a Random Forest model, which captures non-linear relationships, is significantly more effective (Test RMSE: 3.78°C) than a linear model for this task. The analysis confirmed that dew point, latitude, and month are the primary drivers of temperature prediction. Furthermore, our implementation of strategic caching and model persistence proved essential for managing the computational complexity and ensuring the project's successful completion. The resulting model is both accurate and well-understood, providing a powerful tool for meteorological prediction.

Appendix

GitHub link: `https://github.com/sjy13658801014-byte/DSA5208_pj_2`